# Why Use XML for Web Content?

```
<Article>
  <Title>Why Use XML for Web Content?</Title>
  <Author type="primary">
    <Name>Alex Pline</Name>
    <Organization>NASA Headquarters</Organization>
    <Department>Biological and Physical Research</Department>
    <email>alex.pline@nasa.gov</email>
  </Author>
  <Author>
    <Name>Bruce Altner</Name>
    <Organization>NASA Headquarters - SAIC/QSS</Organization>
  </Author>
  <Author>
    <Name>Nathan Shaw</Name>
    <Organization>NASA Headquarters - SAIC</Organization>
  </Author>
  <Author>
    <Name>Colin Enger</Name>
    <Organization>NASA Headquarters</Organization>
  </Author>
  <Location>
    <Web format="xml">
       http://spaceresearch.nasa.gov/general_info/xmldriven.xml
    </Web>
    <Web format="html">
       http://spaceresearch.nasa.gov/general_info/xmldriven.html
    </Web>
    <Web format="pdf">
       http://spaceresearch.nasa.gov/general_info/xmldriven.pdf
    </Web>
  </Location>
</Article>
```

## *Table of Contents*

## *Abstract*

In 2001 when the Office of Biological and Physical Research (OBPR) became NASA's 5th Enterprise, we needed to do a complete makeover of a 4-year old Web site that had become a tangled mess of ugly HTML. We wanted a site that was easier to update, gave us a better way to manage news items, and the ability to publish content to multiple media outlets. XML's open nature, separation of content and presentation, and the ability to easily "process" the content, made it a natural choice.

We developed Document Type Descriptions (DTDs) to model collections of news items and general "web-type" content, as well as Extensible Stylesheet Language (XSL) stylesheets to transform the native XML to HTML via an Extensible Stylesheet Language Transformation (XSLT) processor (SAXON). These transformations were managed using a Java servlet-based administrative interface that also provided tools for adding and updating news items.

In the nearly three years that the site has been live, we have undergone several major site modifications as the content has grown and increased in complexity, without having to modify existing XML content. These updates are only a matter of redesigning the XSL stylesheet(s) and associated graphical elements, and retransforming the XML to HTML. Such changes would have required days, or even months, if we had stayed with an HTML 4.0 mixed content/presentation model. We have also made upgrades to the infrastructure of the administration tool, almost never having to touch any individual pieces of content.

It proved straightforward to include externally authored content to the site, using a DTD written by the Science@NASA organization, a format well suited for "feature article" type content. As the Science@NASA writers transition to producing all of their content in XML, we will use our infrastructure to perform XML to XML transformations to automate the import of Science@NASA content to the NASA Public Web Portal.

The downside of creating native XML content is that the currently available XML authoring and editing tools are still in a primitive stage. Fortunately, the landscape is rapidly changing and there are some promising commercial products on the horizon. It remains to be seen whether these products are implemented using open and accessible standards or proprietary solutions.

In this paper, we discuss the implementation, results and benefits of using this approach for Web content.

## *Introduction*

In 2001 when the Office of Biological and Physical Research (OBPR) became NASA's 5th Enterprise, we needed to do a complete makeover of a 4-year old web site[1] that had become a tangled mess of ugly HTML. Fortunately, our job of dealing with legacy content was minimized since the Enterprise was new and there was a significant structural change in the organization. The Web site required not only a new look, but also virtually all new content. This gave us a clean slate to start with when considering how to build the new site. Developers working on smaller Web sites at NASA Headquarters were starting to become aware of the potential benefits of Java and XML in their work[2,3] and, in fact, at least two pilots were already underway.

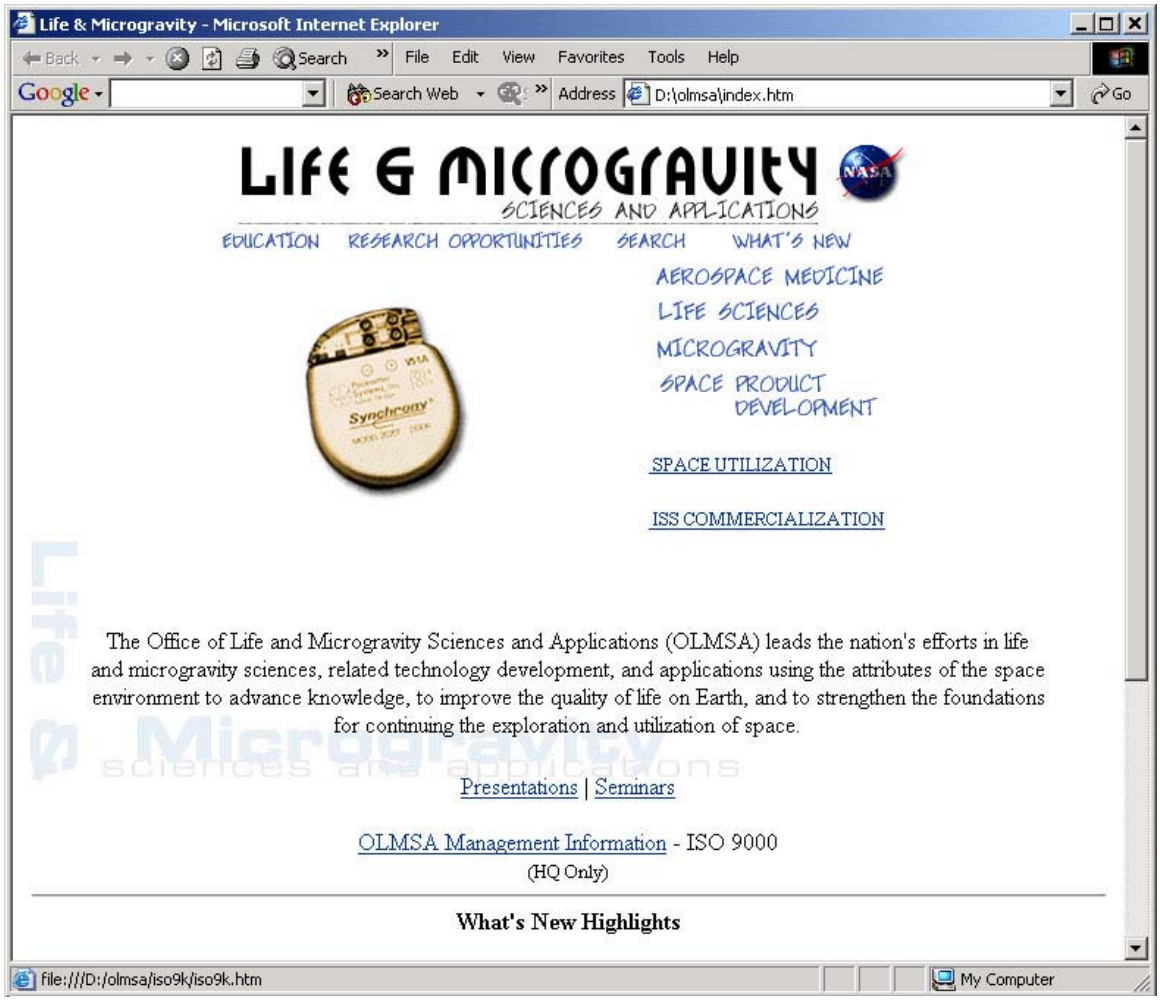


*Figure 1. The Office of Life and Microgravity Science and Applications main page prior to the creation of the Biological and Physical Research Enterprise. Note the mismatch of the graphic fonts, a visible manifestation of the underlying ugliness of the site.*

One of the primary requirements of the new site was that we needed an automated way to mange time sensitive news items to avoid one of the greatest Web *faux pas*, a "What's

New" page that is out of date. This typically happens when items are updated manually. While there are many ways to automate this task, we began to research the benefits of using XML after seeing the RSS-like, XML-based feed[4] of news items from the Science@NASA group[5] at the Marshall Space Flight Center. Since there was no DTD to describe the structure of this XML file, we created one and included some additional elements to suit our needs. As we progressed, we began to see the benefits of using XML to separate content and presentation.

As a temporary solution to the Web site update, we set up a short term HTML "mini-site" to help us refine our graphical and management requirements. Armed with the XML format and a DTD, we developed a method of creating HTML versions of the news information by inserting the news content into HTML templates, using Open Source tools such as Apache Tomcat[6] (a Java servlet container) and JDOM[7] (an API for representation of an XML document as a Java class). The servlet implementation worked very well, exceeded our expectations, and encouraged us to move to an all-XML native site. We continued to extend our use of servlets for transforming XML to HTML as well as other site management tasks.

This project has evolved from the initial idea of using XML for a specific task on a Web site to using XML as the native format for the entire site, embracing the use of XSLT for all aspects of creating and maintaining the Web site. The site and the underlying architecture has been an ongoing and evolving process, with improvements in site architecture and operations being added as requirements arise in a manner that is characteristic of "prototype" or "pilot" program.

In the following sections we describe the overall project requirements, system architecture, content operations and upgrades to the site content and XML administration tools to illustrate the benefits (as well as the downside) of using XML as a native format.

## *Project Requirements*

The basic requirements of the Web site upgrade fell into four main areas:

- Content (the types of data structures);
- Design (the layout of content and graphical elements);
- Distribution of Content (media channels);
- Maintainability (requirements for ease of content changes and site upgrades).

While good maintainability can be achieved with current HTML-based Web site development and administration tools, the advantages that XML offered as a flexible native format coupled with a rapidly growing selection of tools for processing and storing XML-based content made XML an obvious choice.

### Content

One of the major planning tasks that we had to undertake in using XML was the initial modeling of the data structure. Significant planning had to be done *a priori* to insure that

the DTD or schema we created (or adopted) could accommodate all of the types of content anticipated for the site. We also needed to consider how we were going to author content in XML.

**News items**

The minimum set of required data elements for each news item included: title, category, start date, end date, date of the item, associated image and its properties and descriptive text. In order to avoid the "out of date What's New page" problem, and to enable us to enter time-sensitive items well in advance (e.g., announcements of research opportunities) we needed to create a mechanism that allowed for automated revealing and hiding of news items based on their start and end dates. An item would not be shown before its start date and would be archived to an "old news" page after its end date.

**Web Content**

The minimum set of required data elements for "general web content" (the types of structures generally found on web sites) included, not surprisingly, many existing HTML tags, such as paragraphs of text, page headings, tables, lists, images, captions etc. However, we also had several requirements that included container-type data elements such as page heading, title, sections and sub sections, and elements to hold meta data and internal document links. This minimum set had to be expanded later to accommodate other specific types of content, for example, a magazine article. This "richer" set of elements included meta data such as authors, editors and source credit as well as more sophisticated ways of handling images, image credits and captions.

**Content Creation**

At the planning phase of this project, XML native editors were at a primitive stage, requiring the author to work in a tagged environment. We evaluated several software packages and as a result were resigned to working in the tagged environment with a minimum requirement that the author have the ability to preview the content locally in its final transformed state prior to posting on the Web site (or other distribution channel).

## Site Design

The requirements for the design of the site were developed as one would any HTML-based site. We chose to build the site around three sections aimed at specific audiences, each with its own section navigation. However, we wanted to remain as true as possible to the concept of separation of presentation (formatting and navigational elements) and content (data) to insure that any piece of content could be used in any section. This meant no "hardwiring" of section specific information into the XML content. We also wanted the site to have common header and footer elements (text, links and graphics) that needed to be added to each page.

### Distribution of Content

Not only did we intend for the content to be used as HTML on a Web site, but we also wanted the ability to repurpose the content to other distribution channels without having to manually reformat the content for any specific channel. For example, the content might be distributed as a printed document, plain text in an e-mail, or very simple HTML for handheld devices. This also required having the content sans any channel specific information.

### Maintainability

As mentioned above, the maintenance of the news items in an automated way was paramount. Moving items between HTML (or XML) pages manually was not practical. We wanted to be able to enter the specific news item information via a Web form only once, save it as XML, and have the item appear on the appropriate page(s) depending on its date and category properties.

Along with the ability to reuse and redistribute the content without manual reformatting, the ability to maintain or completely change the look and feel (e.g., fonts, color scheme, etc.) and the common elements throughout the entire site, or that of any specific section, without changing any of the XML content files, was a key requirement.

## *System Architecture*

### Content Types and Structure

We created two DTDs; Newsfeed.dtd[8] handles the data requirements for the news items and OBPRWebContent.dtd[9] handles the data requirements of most of the Web content. We picked many of the element names to be similar to their HTML counterparts, a practice that helps understand the role they play, but that also may add to some syntactical confusion when switching between XML and HTML. A complete list of the elements can be found in appendices A and B.

Note that in the Newsfeed DTD the <intro> element, the descriptive text of the item, allows for child elements that are identical to HTML elements, allowing an author to embed HTML formatting in the description of the news item. This gives the author a lot of flexibility in formatting the text of the news item but, on the downside, it has the potential to cause problems when attempting to parse this data if it is not syntactically correct.

Alternately, one could use the XHTML[10] specification if there were no requirements for additional data elements, or a combination of XHTML and application specific elements could be constructed. This may provide some benefit as the XHTML specification will be very familiar to people used to working with HTML and potentially require less training to implement. For example the new xml.nasa.gov site is XHTML compliant.

After attempting to create "feature articles" in the OBPRWebContent.dtd structure, we determined that we did not have enough flexibility in the types of elements. For example there were many pieces of data, both content and meta data, that were associated with more complex content that we might want to process in different ways when transforming from XML to another format. These meta data include author, editor, synopsis, teaser, and sidebar. In the OBPRWebContent.dtd structure, we would have had to treat these as paragraphs of text; without separate elements for these pieces of data we would not be able to include, exclude or specially format them based on the distribution or audience requirements. Instead of developing our own structure, we searched for one that would suit our needs. We found that the Science@NASA organization had developed a DTD to work with their "magazine article" content, so we adopted their DTD for our own article content. Not only did adopting the Science@NASA article DTD solve our problem of needing a more robust DTD for articles, it also facilitated the reuse their content as well. See appendix C for a listing of the elements and their properties in SCI-Story.dtd.

## XSLT Infrastructure

In order to transform the XML to other formats we are using Extensible Style Sheet Language Transformations (XSLT).[11] These transformations use an XSLT processor that parses the XML file and applies a stylesheet written in the Extensible Style Sheet Language (XSL, an XML dialect) to output a variety of formats. The format that is ultimately output is based on the stylesheet and processor. There are many Open Source XSLT processors to choose from for transforming XML, such as those offered by the Apache XML Project[12] (for example, Xalan and FOP). We are currently using the Open Source Saxon XSLT processor, developed by Michael Kay.[13]

In the creation of the HTML for the Web site, several transformations are performed. News items are first transformed to the OBPR Web site format via an XML to XML transformation (from the newsfeed.dtd format to the OBPRWebContent.dtd format). Once all XML files are in this format a second process transforms all XML content to HTML. The transformations are performed automatically via either batch or manual processes, using compiled XSL stylesheets for optimization. Internal tracking of timestamps assures that only those files that have changed since their last transformation are processed.

### News Items

Initially we did not use XSLT for the news items. Instead, the HTML pages were rendered using compiled Java code (templates) that contained the HTML markup embedded in System.out.println() statements. This crude approach posed several problems and proved to be too difficult to maintain. We subsequently switched to XSLT transformations, as described above, in which an intermediate XML to XML transformation is done to recast the news items as "standard" site XML files valid to the OBPRWebContent.dtd format. After this is done, these XML files are transformed to HTML along with the rest of the site content.

Transitioning from the template to the XSLT approach for the news items raised an interesting problem. As mentioned in the Content Types and Structure section above, the <intro> element for each news item could potentially contain embedded HTML. If the HTML markup is entered incorrectly the news item will not be well-formed XHTML and the transformation will abort. We therefore had to add a step in the processing to check for well-formed XHTML using the Open Source program JTidy[14] (a Java version of Tidy). This step throws an exception when the transformation fails, bringing up another form through which the author may chose to accept Tidy's corrections or to correct the mistakes manually.

During the XML to XML news item transformation a variety of OBPRWebContent.dtd formatted XML files are produced, based on the category and date elements in each news item. This provides a mechanism for creating separate HTML files on the Web site for each category of current news items (including an "all items" page) as well as archive files of news items separated by year of publication. Currently we have more than 300 news items spread over 6 categories and 3 years. It would a be very menial and time consuming job to manually move items between files as time progresses. Having the ability to enter the content once and automate this process based on the meta data for each news item, saves a tremendous amount of manual effort and assures that the news pages will always be current. Below is a typical news page. Note the internal subnavigation defined in the OBPRWebContent.dtd format allows for both links and Javascript dropdown menu. All of the HTML code to accomplish this is created during the XSLT process.


*Figure 2. A typical news page on spaceresearch.nasa.gov showing the category dropdown and archive menus that select the various news pages. All of these HTML elements and pages are generated via the XSLT process.*

Another file that is created during the news item XML to XML transformation is a “top 5” version of the newsfeed XML file. One the main page of the site a Flash application displays the top 5 news items. One of the benefits of XML and Flash (v. 5 or higher) is that the Flash application will read the XML directly. Once again the news information, displayed in another format in a different location, is assured to be in sync with the rest of the site since they share the same source file. Note that the main page is the only native HTML page on the site. This was a conscious choice: there is only one instance of this page and it would be even more work to maintain a separate XSL stylesheet, and its associated transformation code, than to maintain the HTML directly.

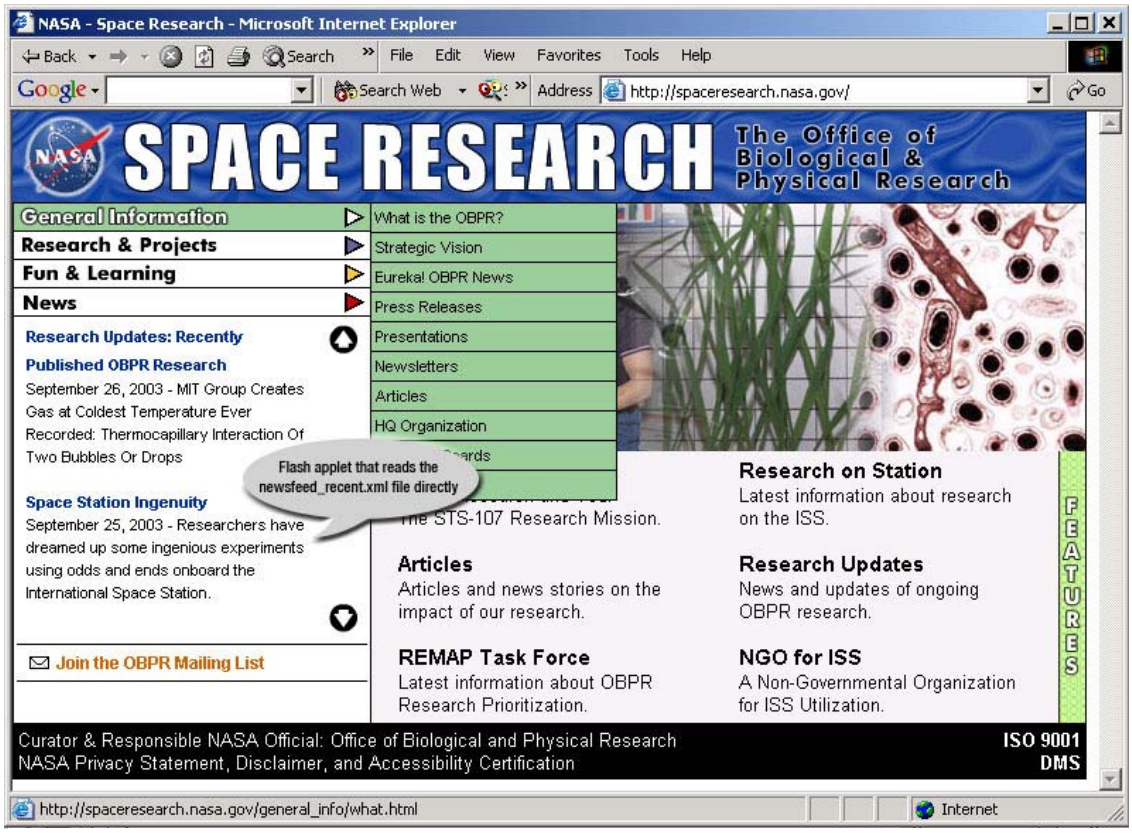


*Figure 3.The spaceresearch.nasa.gov main page showing the Flash application news scroller.*

**Main Site Content**

Initially our achitecture included an XSL stylesheet for each of the three main section of the site ("General Information", "Research & Projects", and "Fun & Learning"), in which the stylesheet added the common site elements (header, footer etc.) as well as the specific navigation for that section. There were several problems with this architecture. While using XSLT to create HTML reduced the places we needed to make changes to the three individual XSL stylesheets, we were still having to make modifications to other files where the section navigation show up, for example, arrays in Javascript files for dropdown menus in each section. Also, if we wanted to expand the number of sections, we would have to create another XSL stylesheet.

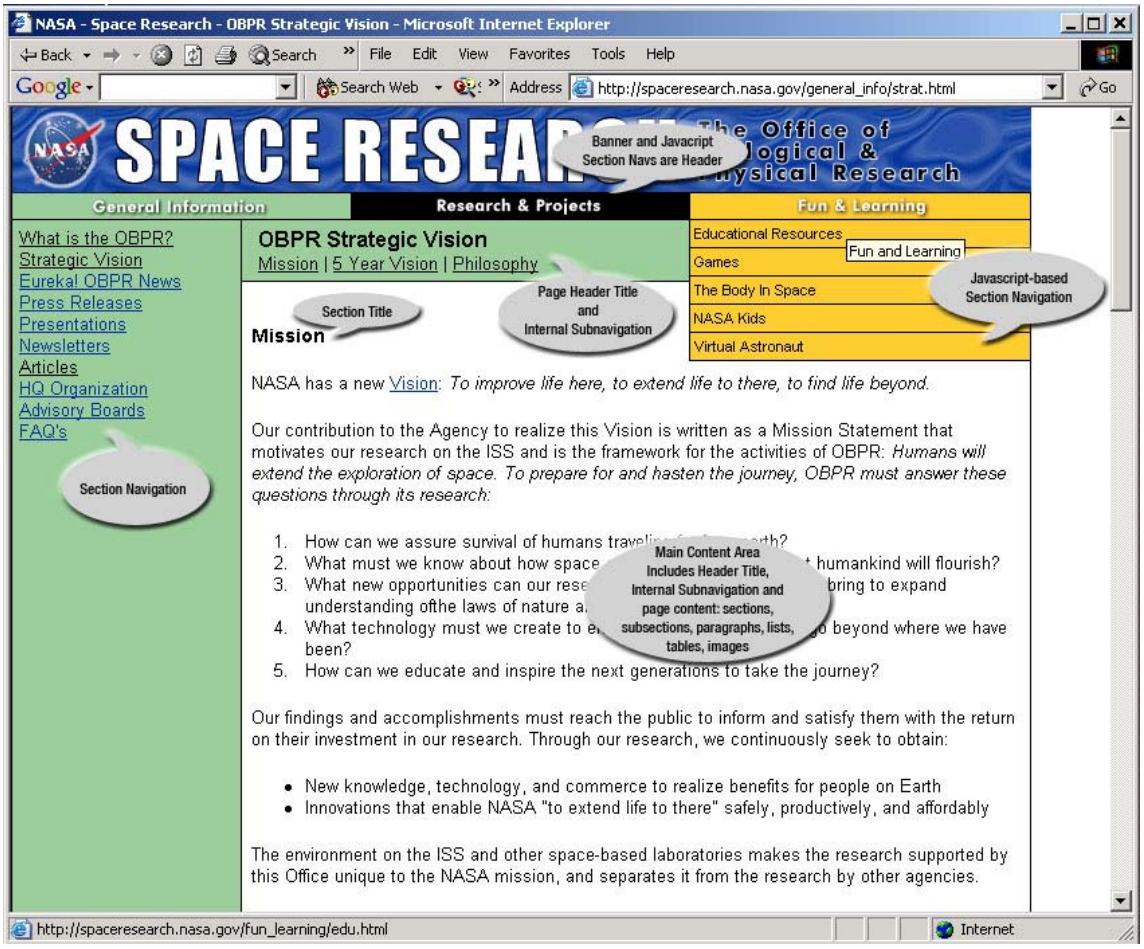


*Figure 4. A typical page on spaceresearch.nasa.gov showing the various elements that are assembled during the XSLT process.*

These issues drove us to create a much more modular XSLT infrastructure for many of the components that are assembled in the XSLT process. By doing this, all of the "source" information used to create the HTML files that was previously hard coded into the XSL stylesheet, either in individual section stylesheets or in multiple places in a

single stylesheet, could now be placed in one set of configuration files used in the XSLT process. This included a master XML configuration file containing all the relevant information for each section and various XML (actually XHTML) files containing snippets of code for the header, footer, each section navigation and top navigation used in each section. This allowed us to create a single general XSL stylesheet, which would include these code snippets based on the information in the configuration file and apply them during the transformation. The XSLT also created the appropriate Javascript arrays containing the section navigation items. These Javascript arrays are used in every page, including the Web site's main page. Below is a snippet of the XML configuration file showing the locations of the various XHTML objects that are assembled during the XSLT process:

```
<configuration>
<!-- general_info  -->
   <directory name="general_info">
            <header>../../common/headers/OBPRHeader.xml</header>
            <css>/common/css/styles_general_info.css</css>
            <iecss>/common/css/iestyles_general_info.css</iecss>
            <topnav>../../common/topnavs/TopNav_general_info.xm/topnav>
            <sectionnav>../../common/sectionnavs/SectionNav_general_info.xml</sectionnav>
            <footer>../../common/footers/OBPRFooter.xml</footer>
            <primarycolor>#99cc99</primarycolor>
            <secondarycolor>#ffffff</secondarycolor>
   </directory>
</configuration>
```

This indicates that navigation items for the "General Information" section are controlled by elements in the SectionNav_general_info.xml XHTML file:

```
<td bgcolor="#99cc99" width="160">
    <table cellspacing="5" cellpadding="0" border="0">
        <tr>
            <td>
                <a href="/general_info/what.html">What is the OBPR?</a> <br />
                <a href="/general_info/strat.html">Strategic Vision</a>  <br />
                <a href="/news/news.html">Eureka! OBPR News</a>  <br />
                <a href="/general_info/pressrel.html">Press Releases</a>  <br />
                <a href="/general_info/prespublic.html">Presentations</a>  <br />
                <a href="/general_info/prespublic.html#newsletters">Newsletters</a>  <br />
                <a href="/general_info/art.html">Articles</a>  <br />
                <a href="/general_info/org.html">HQ Organization</a>  <br />
                <a href="/general_info/adv.html">Advisory Boards</a>  <br />
                <a href="/general_info/faq.html">FAQ's</a><br />
                <img src="/common/images/shim.gif" width="160" height="1" border="0" alt="" />
            </td>
        </tr>
    </table>
</td>
```

As a result, we only need to change information in a single place to affect changes throughout the entire site. This includes, adding new major sections, section navigation items, header, footer and section colors. Once the change is made, all the XML files are transformed and the site HTML is updated.

The details of the main XSL stylesheet are available on the Web site.[15] In a further attempt to modularize, this stylesheet references other stylesheets, located at dtd.nasa.gov, which handle date/time, string, and file information issues.

## Application Development

### Java Framework

The synergy between Java and XML has been widely discussed.[16] It has been said that Java provides "portable code" while XML provides "portable data." We recognized this close connection very early in the design process and decided to base our implementation on J2EE technology, in particular, Java servlets. Servlets provided a robust environment in which to develop Web-based applications and the ready availability of mature Java APIs for XML validation, parsing, and transforming greatly simplified the task at hand.

Using Java servlets in an application layer required a "servlet container" (engine). There are many servlet containers available today. Many of these are bundled with expensive middleware application server products, such as IBM's WebSphere and BEA System's WebLogic. We chose the Tomcat servlet container, released under the Apache Software License and used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. In doing so we were assured of having a high quality, low cost (free), thoroughly tested, up-to-date platform upon which we could build our application.

When we began this project we deployed Tomcat 3.2. At the time of this writing were are using Tomcat 4.1.27 and will likely upgrade to Tomcat 5.0.x soon after its release. With each new version, Tomcat improves upon prior versions in terms of security, performance, remote access tools, scalability and reliability, integration with the operating system, and session handling, to name just a few. Tomcat 4 implemented the Servlet 2.3 and JavaServer Pages 1.2, and included many additional features that make it a useful platform for developing and deploying web applications and web services. Tomcat 5 implements the Servlet 2.4 and JavaServer Pages 2.0 specifications.

As mentioned, servlets provided a secure and stable platform for building server-based Web applications. Because servlets are Java technology all of the Java class libraries that have been developed for XML processing were immediately available for use with little more effort than downloading the library and placing it in the proper Tomcat directory. In our work to date the most important of these libraries have been JDOM for accessing, manipulating, and outputting XML data, Saxon for XSLT processing, Ant (Apache), a Java application build tool, Xerces (Apache) for XML parsing, JTidy for XHTML validation, JavaMail (Sun) for e-mail messaging, and Velocity (Apache) for JSP-like template processing.

### OBPR XML Administration Tool Operations

All of the operations of the XML Administration tool running within the Tomcat framework are performed from a Web-based administration panel on a Unix server. This box also acts as our site staging server for previewing changes in a protected environment. All transformations run daily as a cron job so that XML content added to the server will update the site automatically. These processes can be run manually as well, in case there is a need to add or change content immediately.

On the main administration page there are options to manage the news items, perform manual transformations (news items, Web content or both) and preview the entire site or individual XML/HTML pages. This interface makes the tool incredibly easy to use. Although we usually use FTP to transfer files to the server, a file upload option was added for transferring files when FTP is not an option, for example when using the NASA HQ Secure Nomadic Access application for remote access through the firewall.

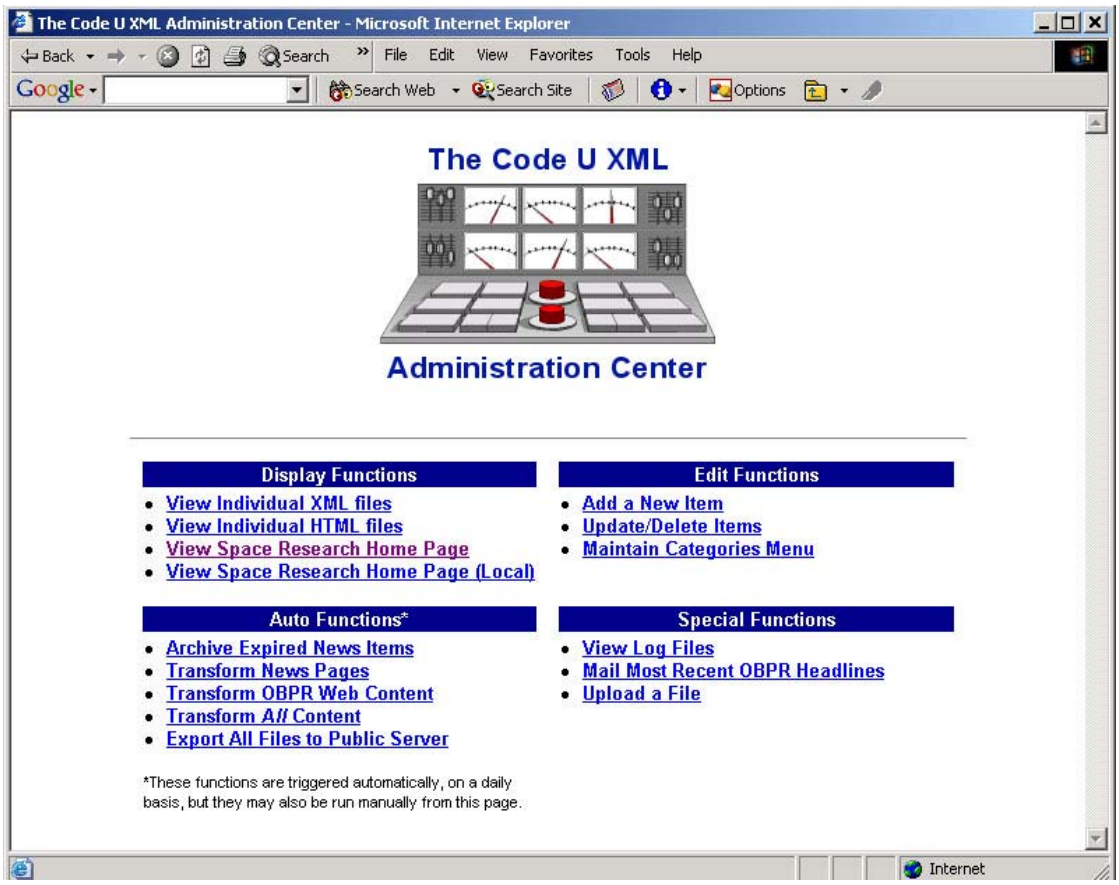


*Figure 5. The OBPR XML Administration Page used to manage news items and XSLT transformations*

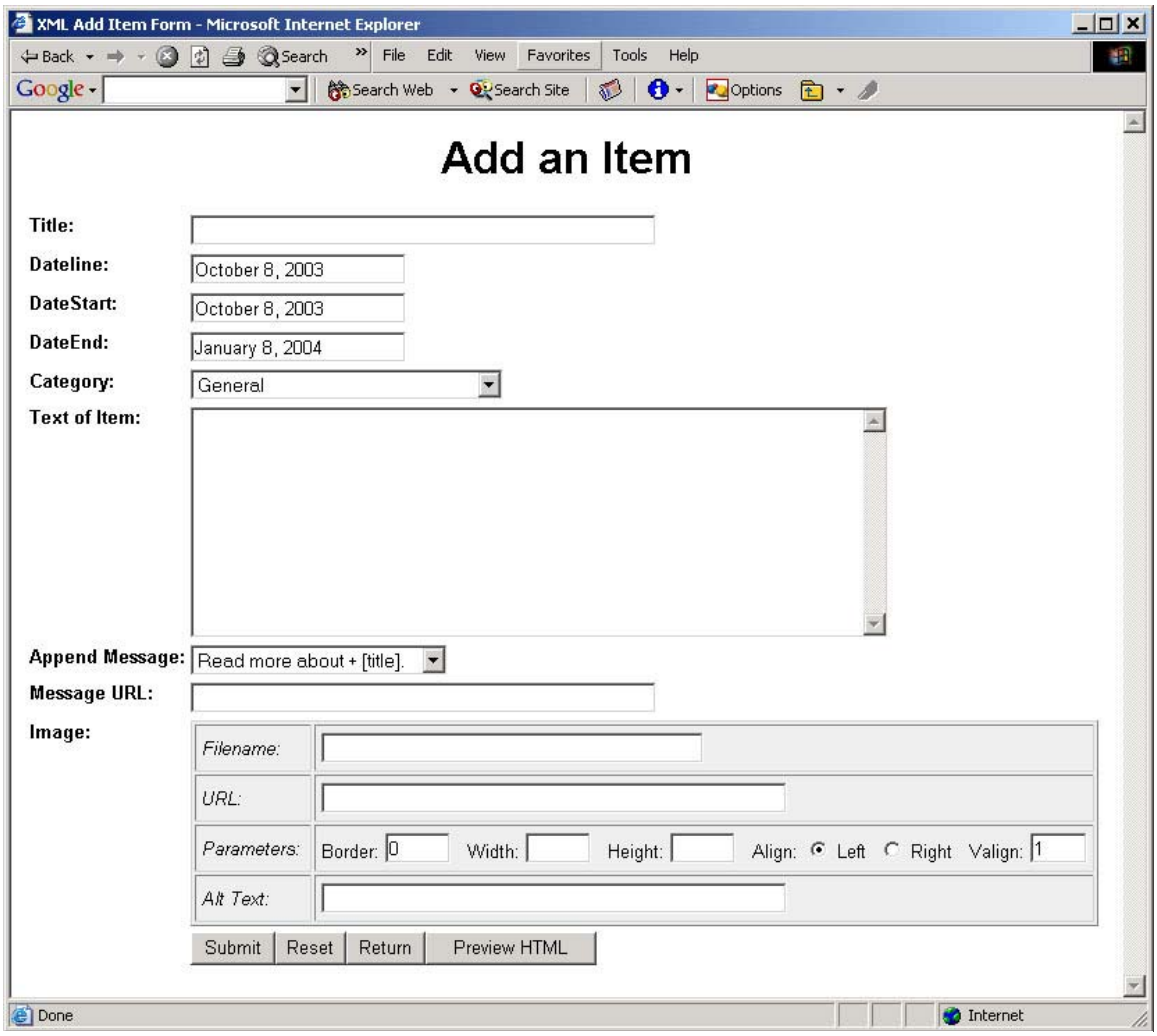


*Figure 6. Adding a news item is as easy as filling out this web form. Dave validation is performed in the background and Tidy performs XHTML validation on the "Text of Item" field when the item is submitted. The system enters this data into the newsfeed.xml file. The "Preview HTML" button allows the author to see how the item will appear when rendered as HTML.*

## *Examples of Site Changes and Upgrades*

In the three years that the site has been operational, we have made a number of site changes that have been very easy to implement given the native XML format and XSLT infrastructure. We routinely add or change the section navigation and update header/footer links. To date, we have added one new major section. The new section was for STS-107, an OBPR research mission. We were able to easily add a custom header and section navigation and assign colors, all through the configuration files.

When OBPR created an e-mail list for distribution of news items and announcements, it proved easy to transform the news items from XML to stylized text that includes hyperlinks pointing to the full news items on the Web site. The default item text can be

edited in a Web form and previewed prior to e-mailing to the list. The system simultaneously produces an RSS feed containing the news items that were e-mailed to the list, providing a second method for distributing this content. This feed is syndicated through several RSS collection sites, such as Syndic8.com.


*Figure 7. The STS-107section added midway through the site's life. Note the custom header image.*

Another recent addition to the site was a "printer friendly" format of all site pages. By developing new XSL stylesheet (OBPRWebContentPrint.xsl),[17] we are able to produce HTML pages that print gracefully through the browser. A small change to the DTD was made to add an attribute to any inline link to specify whether the printer friendly XSL should display the link URL. After this XSL was developed the OBPRWebContent.xsl stylesheet was changed to add a link on each page to the printer friendly version. The entire site was then transformed and updated automatically. "Printer friendly" pages are now automatically produced for all site content.

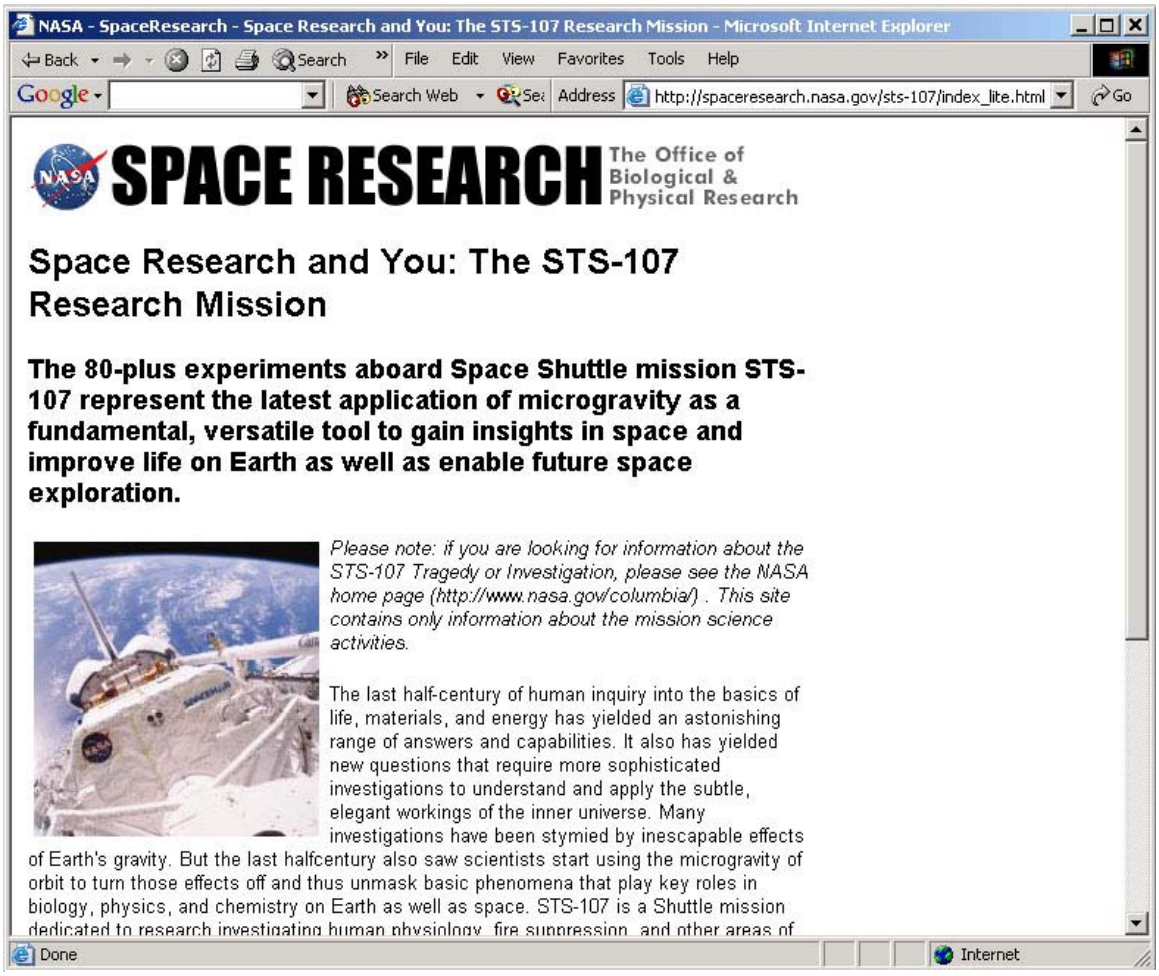


*Figure 8. The "printer friendly" version of the STS-107 main page. Note that no navigational elements or other extraneous web-only information shows up in this version. The content width is constrained so that the page will print gracefully.*

We have been experimenting informally with using XSL Formatting Objects (XSL-FO)[11] and an Open Source XSL-FO processor from Apache called FOP for producing PDF files from our XML content. This experiment has been only moderately successful due primarily to the limitations of this Open Source tool. There are a number of upcoming tools, both Open Source and commercial products, that are addressing the shortcomings of XSL-FO and we expect that we will achieve greater success with this in the near future.

**Science@NASA Content and the NASA Public Web Portal**

When we adopted the Science@NASA format (SCI-Story.dtd)[18] for our "feature article" content, there were two motivations. One, the format fits the OBPR article content better than the OBPRWebContent.dtd format, and two, Science@NASA develops a significant amount of content for OBPR and we wanted to be able to easily repurpose this content on our site instead of linking to their standard publications on science.nasa.gov. We developed two XSL stylesheets, OBPRArticles.xsl[19] and OBPRArticlesPrint.xsl,[20]

normal and “printer-friendly,” respectively, for use with our XSLT infrastructure. With these stylesheets we can automatically transform the Science@NASA articles in a manner analogous to the way we transform Web content using the OBPRWebContent.xsl styleshheets.

When the NASA Public Web Portal (www.nasa.gov) came online, we immediately began investigating how to supply content directly via XML rather than entering it through the Portal content management system (CMS). The Portal project has published an XML schema[21] for importing content into the CMS. The overview of the schema shows elements for meta data and one large CDATA section for the content body. This content body is HTML that must be formatted according to the Portal style guidelines.

Since the Science@NASA content is now fully developed in XML, and we have a significant amount of experience with both the SCI-Story DTD and the Portal schema, OBPR has offered our XSLT infrastructure to transform the Science@NASA content to the Portal Import format on an ongoing basis. We have developed an XSL stylesheet (OBPRArticlesPortalDetail.xsl)[22] that maps the elements in the SCI-Story format to meta data elements in the Portal Import format and transforms the body content into HTML that conforms to the Portal style guidelines. It is planned to automate this transformation and subsequent importing into the CMS. To date, this transformation is in the testing phase and will soon be put into production. For comparison, it takes a person familiar with the CMS 2-3 hours to reformat HTML from science.nasa.gov for the Portal, whereas our approach is fully automatic. Science@NASA publishes about 3 articles per week, so the use of this transformation saves about 6-9 person hours per week and is implemented in a significantly more accurate and consistent manner.

## *Conclusion*

### Benefits of XML as a Native Format

The basic tenant of XML, separation of content and presentation, has produced significant efficiencies in the way we manage and maintain the OBPR Web site. Beyond the savings in human maintenance time, using XML has allowed us to consolidate the portions of source information that comprise the site into canonical locations which can be read, processed, displayed and distributed in a variety of ways. Once this data is in an “available” format, the information is truly “extensible” and will result in additional efficiencies as new uses are developed, so the benefits go far beyond the first order goal of increasing Web site maintainability. While, there is a significant learning curve in the development of an XSLT infrastructure, this kind of approach produces a significant return on investment through savings on maintenance costs, better data quality and currency, and a greater number of potential uses for the XML-based data.

### Continuing Problem: WYSIWYG XML Editing

When starting this project, we knew that tools for creating well-formed and valid XML in a familiar (e.g. WYSIWYG) environment would be a challenge. This is currently the Achilles heel of creating XML content. At that time, the best tool we could find was

called XML Writer, a well designed “tagged environment” editor that also had the capability of performing local XSLT transformations.

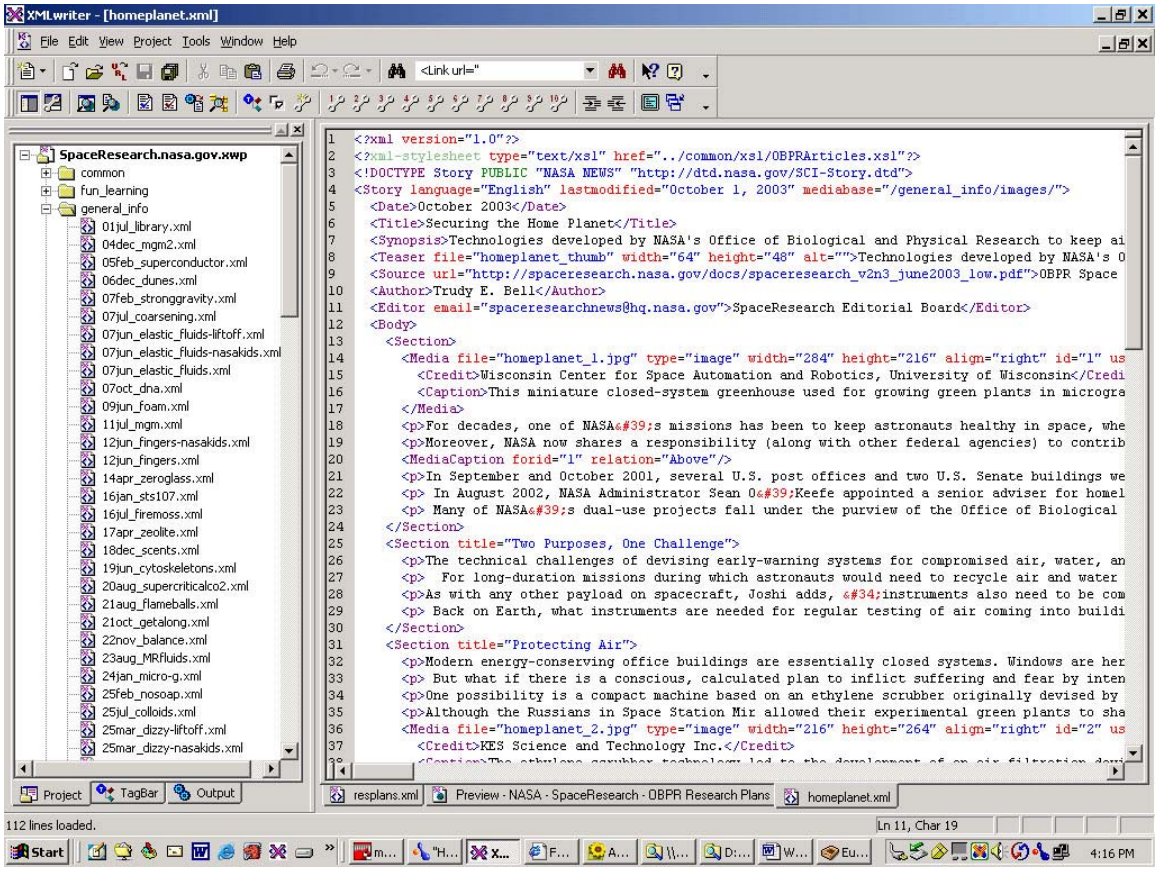


*Figure 9. An XML Writer screen shot. It makes working in a tagged environment easier than Notepad, offers configurability for custom XSLT processors and project management, however is not appropriate for not technical users.*

While this application is certainly appropriate for technical users, it is not appropriate for the average non-technical content creator used to working in applications such as Microsoft Word. We have looked into various Microsoft Word add-ons[23] which output XML by constraining users to various Word styles and mapping those styles to XML elements. Evaluations of the products show that they are quite limited and often require a significant amount of customization of Word, depending on the structure of the XML. Since our XML is “semi-structured” in that there is a large variability in kinds and numbers of element, this was especially problematic. We did not believe that the benefit of customizing Word with these plug-ins was worth the effort, especially since our use of XML is currently limited to the Web application. We have also conducted extensive analyses of the new Office 2003 suite,[24] which provides the most XML-aware versions of the Office products yet. However, based on our evaluation of the beta product, using our “semi-structured” types of content, it appears that Word 2003 will still require some assistance from a plug-in.

As XML becomes more prevalent in our organization we will have to reevaluate the available products. The field is rapidly changing and new products show great promise in helping to solve this problem. Companies that are producing stand-alone XML editors, such as Arbortext and Adobe, have promising products of various levels of complexity and sophistication.

## *References and Resources*

---

[1] Space Research, The Office of Biological and Physical Research: http://spaceresearch.nasa.gov

[2] Recommendation for Java Servlet Implementation for the NASA Earth Science Enterprise, SAIC ISEM Contract, Service Request 2000-0000714

[3] Recommendation for XML Implementation for the NASA Earth Science Enterprise, SAIC ISEM Contract, Service Request 2000-0000726

[4] Science@NASA XML Newsfeed: http://liftoff.msfc.nasa.gov/Content.xml

[5] Science@NASA: http://science.nasa.gov

[6] Apache Tomcat: http://jakarta.apache.org/tomcat/

[7] JDOM Project: http://www.jdom.org

[8] OBPR Newsfeed DTD: http://dtd.nasa.gov/Newsfeed.dtd

[9] OBPR Web Content DTD: http://dtd.nasa.gov/OBPRWebContent.dtd

[10] XHTML 1.0 Specification: http://www.w3.org/TR/xhtml1/

[11] The Extensible Stylesheet Language Family (XSL) Specifications (XSL/XSLT): http://www.w3.org/Style/XSL/

[12] Apache XML Project: http://xml.apache.org/

[13] Saxon XSLT processor: http://saxon.sourceforge.net

[14] JTidy Project: http://sourceforge.net/projects/jtidy

[15] OBPR Web Content XSL: http://spaceresearch.nasa.gov/common/xsl/OBPRWebContent.xsl

[16] Java and XML Synergy: http://java.sun.com/xml/ncfocus.html#synergy

[17] OBPR Web Content XSL (printer friendly pages): http://spaceresearch.nasa.gov/common/xsl/OBPRWebContentPrint.xsl

[18] Science@NASA DTD: http://dtd.nasa.gov/SCI-Story.dtd

[19] OBPR Articles XSL: http://spaceresearch.nasa.gov/common/xsl/OBPRArticles.xsl

[20] OBPR Articles XSL (printer friendly pages): http://spaceresearch.nasa.gov/common/xsl/OBPRArticlesPrint.xsl

[21] Portal Import schema: http://dtd.nasa.gov/content_v2.0.xsd

[22] SCI-Story to Portal Import schema XSL: http://spaceresearch.nasa.gov/common/xsl/OBPRPortalDetail.xsl

[23] One Button NASA Publishing at NASA HQ, Nathan Shaw: http://webwork.larc.nasa.gov/doc_2000/XML_NASA1ButtonPublishing_060201.ppt

[24] Office 2003 Evaluation for the Office of Biological and Physical Research, NASA/SAIC ISEM Team

## *Appendices*

The following appendices are included for ease of reference. However, they may not be the latest versions. Please see the links in the References and Resources section for the latest versions.

## Appendix A: newsfeed.dtd

Version 1.2 - 07/15/2002

```
<!ELEMENT newsfeed (channel+)>
<!ELEMENT channel (item+)>
        <!ATTLIST channel
                id NMTOKEN #REQUIRED
                name CDATA #REQUIRED
                url CDATA #REQUIRED>
<!ELEMENT item (title | date | dateStart? | dateEnd? | intro | category? | href? | source? | image? | keywords?)*>
        <!ATTLIST item ID NMTOKEN #REQUIRED>

<!ELEMENT title (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT dateStart (#PCDATA)>
<!ELEMENT dateEnd (#PCDATA)>
<!ELEMENT intro (#PCDATA | em | a | br | p | ul | ol | strong | pre | b | i )*>
<!ELEMENT category (#PCDATA)>
<!ELEMENT href (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT image (name | url | alt? | border? | width? | height? | align? | valign?)*>
<!ELEMENT name ( #PCDATA ) >
<!ELEMENT url ( #PCDATA ) >
<!ELEMENT alt ( #PCDATA ) >
<!ELEMENT border ( #PCDATA ) >
<!ELEMENT width ( #PCDATA ) >
<!ELEMENT height ( #PCDATA ) >
<!ELEMENT align ( #PCDATA ) >
<!ELEMENT valign ( #PCDATA ) >
<!ELEMENT keywords (#PCDATA)>
<!ELEMENT crossReference (#PCDATA)>

<!-- commom HTML elements -->
<!ELEMENT a (#PCDATA | em | br | img)*>
        <!ATTLIST a
                class   CDATA           #IMPLIED
                href    CDATA           #IMPLIED
                target  CDATA           #IMPLIED
                name    CDATA           #IMPLIED>
<!ELEMENT br (#PCDATA)>
        <!ATTLIST br
                clear   CDATA           #IMPLIED>
<!ELEMENT strong (#PCDATA | em | a | br)*>
<!ELEMENT em (#PCDATA | strong | a | br)*>
<!ELEMENT b (#PCDATA | strong | a | br)*>
<!ELEMENT i (#PCDATA | strong | a | br)*>
<!ELEMENT p (#PCDATA | strong | em | a | br | Image )*>
<!ELEMENT pre (#PCDATA)>
```

```
<!-- img -->
<!ELEMENT img (#PCDATA)>
        <!ATTLIST img
                name    CDATA           #IMPLIED
                src     CDATA           #REQUIRED
                alt     CDATA           #REQUIRED
                border  CDATA           #IMPLIED
                width   CDATA           #IMPLIED
                height  CDATA           #IMPLIED
                align   CDATA           #IMPLIED
                valign  CDATA           #IMPLIED>

<!-- Unordered List -->
<!ELEMENT ul (li*)>
         <!ATTLIST ul
                Title   CDATA           #IMPLIED
                URL     CDATA           #IMPLIED
                Type    CDATA           "Unordered">

<!-- Ordered List -->
<!ELEMENT ol (li*)>
  <!ATTLIST ol
                Title   CDATA           #IMPLIED
                URL     CDATA           #IMPLIED
                Type    CDATA           "Unordered">

<!-- List Elements-->
<!ELEMENT li (#PCDATA | a | br | ul | ol)*>
  <!ATTLIST li
                URL CDATA #IMPLIED
                style CDATA #IMPLIED
                Caption CDATA #IMPLIED>
```

## Appendix B: OBPRWebContent.dtd

Version 2.9 - 07/29/2003

```
<!ELEMENT WebContent (Audiences?, Navigation?, Content+)>
<!-- common HTML elements -->
<!ELEMENT a (#PCDATA | em | a | br)*>
<!ATTLIST a
        class CDATA #IMPLIED
        href CDATA #IMPLIED
        target CDATA #IMPLIED
        name CDATA #IMPLIED
        printURL (yes | no) "no"
>
<!ELEMENT br (#PCDATA)>
<!ATTLIST br
        clear (all) #IMPLIED
>
<!ELEMENT strong (#PCDATA | em | a | br)*>
<!ELEMENT em (#PCDATA | strong | a | br)*>
<!ELEMENT p (#PCDATA | strong | em | a | br | Image)*>
```

```
<!ELEMENT pre (#PCDATA)>
<!-- List -->
<!ELEMENT List (ListItem*)>
<!ATTLIST List
        Title CDATA #IMPLIED
        URL CDATA #IMPLIED
        Type (Unordered | Ordered) "Unordered"
>
<!ELEMENT ListItem (#PCDATA | strong | em | a)*>
<!ATTLIST ListItem
        URL CDATA #IMPLIED
        Caption CDATA #IMPLIED
>
<!-- DataTable -->
<!ELEMENT DataTable (DataTableHeader, DataTableRow+)>
<!ATTLIST DataTable
        Title CDATA #IMPLIED
        Summary CDATA #IMPLIED
        Width CDATA #IMPLIED
        Height CDATA #IMPLIED
        Border CDATA "0"
        CellSpacing CDATA "0"
        CellPadding CDATA "0"
        Align (left | center | right) #IMPLIED
        BGColor CDATA #IMPLIED
>
<!ELEMENT DataTableHeader (DataTableColumnHeader+)>
<!ELEMENT DataTableColumnHeader (#PCDATA)>
<!ATTLIST DataTableColumnHeader
        URL CDATA #IMPLIED
        RowSpan CDATA #IMPLIED
        ColSpan CDATA #IMPLIED
        Align (left | center | right) #IMPLIED
        VAlign (top | middle | bottom) #IMPLIED
        BGColor CDATA #IMPLIED
        Width CDATA #IMPLIED
        Height CDATA #IMPLIED
>
<!ELEMENT DataTableRow (DataTableRowHeader*, DataTableRowCell+)>
<!ELEMENT DataTableRowHeader (#PCDATA)>
<!ATTLIST DataTableRowHeader
        URL CDATA #IMPLIED
        RowSpan CDATA #IMPLIED
        ColSpan CDATA #IMPLIED
        Align (left | center | right) #IMPLIED
        VAlign (top | middle | bottom) #IMPLIED
        BGColor CDATA #IMPLIED
        Width CDATA #IMPLIED
        Height CDATA #IMPLIED
>
<!ELEMENT DataTableRowCell (#PCDATA)>
<!ATTLIST DataTableRowCell
        URL CDATA #IMPLIED
        RowSpan CDATA #IMPLIED
        ColSpan CDATA #IMPLIED
        Align (left | center | right) #IMPLIED
```

```
        VAlign (top | middle | bottom) #IMPLIED
        BGColor CDATA #IMPLIED
        Width CDATA #IMPLIED
        Height CDATA #IMPLIED
>
<!-- Image -->
<!ELEMENT Image EMPTY>
<!ATTLIST Image
        SourceURL CDATA #REQUIRED
        LinkURL CDATA #IMPLIED
        Type (block | inline) "block"
        AltText CDATA #IMPLIED
        LongDesc CDATA #IMPLIED
        Width CDATA #REQUIRED
        Height CDATA #REQUIRED
        Align (left | center | right) #IMPLIED
        Name CDATA #IMPLIED
        Caption CDATA #IMPLIED
        CaptionAlign (top | left | right | bottom) "bottom"
        hspace CDATA "0"
        Border CDATA "0"
>
<!-- Navigation -->
<!ELEMENT Navigation (MainMenu?, SubMenu?, DropDownMenu?)>
<!ELEMENT MainMenu (MainMenuItem*)>
<!ELEMENT MainMenuItem (#PCDATA)>
<!ATTLIST MainMenuItem
        URL CDATA #REQUIRED
>
<!ELEMENT SubMenu (SubMenuItem | Image)*>
<!ELEMENT SubMenuItem (#PCDATA)>
<!ATTLIST SubMenuItem
        URL CDATA #REQUIRED
>
<!ELEMENT DropDownMenu (DropDownMenuItem+)>
<!ELEMENT DropDownMenuItem (#PCDATA)>
<!ATTLIST DropDownMenuItem
        URL CDATA #REQUIRED
>
<!-- Audiences -->
<!ELEMENT Audiences (Audience+)>
<!ELEMENT Audience (#PCDATA)>
<!-- Content containers -->
<!ELEMENT Content (Header+, Section*)*>
<!ELEMENT Header EMPTY>
<!ATTLIST Header
        Title CDATA #REQUIRED
>
<!ELEMENT Section (#PCDATA | strong | em | a | p | br | pre | List | DataTable
| Image | SubSection)*>
<!ATTLIST Section
        Title CDATA #IMPLIED
        URL CDATA #IMPLIED
        Anchor CDATA #IMPLIED
>
<!ELEMENT SubSection (#PCDATA | strong | em | a | p | br | pre | List |
```

```
DataTable | Image)*>
<!ATTLIST SubSection
        Title CDATA #IMPLIED
        URL CDATA #IMPLIED
        Anchor CDATA #IMPLIED
>
```

## Appendix C: SCI-Story.dtd

Version 2.9 - 06/18/2003

```
<!ELEMENT Story (Date, Title, Synopsis, Teaser, Source*, Author+, Editor*, Translator*, ChangeLog*, Body, Extra*)>
<!ATTLIST Story
        language (English | Spanish) "English"
        lastmodified CDATA #REQUIRED
        mediabase CDATA #REQUIRED
        pulldate CDATA #IMPLIED
>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Synopsis (#PCDATA)>
<!ELEMENT Teaser (#PCDATA)>
<!ATTLIST Teaser
        file CDATA #REQUIRED
        width CDATA #REQUIRED
        height CDATA #REQUIRED
        alt CDATA #REQUIRED
        leadout CDATA #IMPLIED
>
<!ELEMENT Author (#PCDATA)>
<!ATTLIST Author
        email CDATA #IMPLIED
>
<!ELEMENT Editor (#PCDATA)>
<!ATTLIST Editor
        email CDATA #IMPLIED
>
<!ELEMENT Translator (#PCDATA)>
<!ATTLIST Translator
        email CDATA #IMPLIED
>
<!ELEMENT ChangeLog (Change+)>
<!ELEMENT Change (#PCDATA)>
<!ATTLIST Change
        date CDATA #REQUIRED
>
<!ELEMENT Classification (Category*, Index*)>
<!ATTLIST Classification
        label CDATA #REQUIRED
>
<!ELEMENT Category (#PCDATA)>
<!ELEMENT Index (#PCDATA)>
<!ATTLIST Index
        keywords CDATA #REQUIRED
>
<!ELEMENT Body (Section+)>
```

```
<!ELEMENT Section (Media | MediaCaption | Table | Sidebar | ul | ol | p)*>
<!ATTLIST Section
	title CDATA #IMPLIED
>
<!ELEMENT Sidebar (Media | MediaCaption | Table | ul | ol | p)*>
<!ATTLIST Sidebar
	title CDATA #IMPLIED
	align (left | right | center | auto) "auto"
	use (content | related) "content"
>
<!ELEMENT Media (Link?, Credit?, Source?, Caption?)>
<!ATTLIST Media
	file CDATA #REQUIRED
	width CDATA #REQUIRED
	height CDATA #REQUIRED
	type CDATA #IMPLIED
	align (left | right | center | auto) "auto"
	alt CDATA #REQUIRED
	id CDATA #IMPLIED
	use (content | related | decoration) "content"
>
<!ELEMENT Credit (#PCDATA)>
<!ATTLIST Credit
	url CDATA #IMPLIED
>
<!ELEMENT Caption (#PCDATA | Link | b | i | sub | sup)*>
<!ELEMENT MediaCaption EMPTY>
<!ATTLIST MediaCaption
	forid CDATA #REQUIRED
	relation CDATA #REQUIRED
>
<!ELEMENT Link (#PCDATA | b | i | sub | sup)*>
<!ATTLIST Link
	url CDATA #REQUIRED
	nasa (true | false) #REQUIRED
	type (document | image | movie) #IMPLIED
	width CDATA #IMPLIED
	height CDATA #IMPLIED
>
<!ELEMENT Source (#PCDATA)>
<!ATTLIST Source
	url CDATA #IMPLIED
>
<!ELEMENT Table (tr+)>
<!ATTLIST Table
	align (left | right | center | auto) "auto"
	style (plain | row_highlight | column_highlight | layout | layout_highlight1 | layout_hightlight2) "plain"
>
<!ELEMENT tr (th | td)*>
<!ATTLIST tr
	height CDATA #IMPLIED
>
<!ELEMENT th (#PCDATA)>
<!ATTLIST th
	colspan CDATA #IMPLIED
	width CDATA #IMPLIED
```

```
        rowspan CDATA #IMPLIED
>
<!ELEMENT td (#PCDATA | Media | MediaCaption | p)*>
<!ATTLIST td
        colspan CDATA #IMPLIED
        width CDATA #IMPLIED
        rowspan CDATA #IMPLIED
>
<!ELEMENT Extra (Audio?, Editorial?, WebLinks?, Index?, Glossary?, Pronunciations?, Translation*)>
<!ELEMENT Audio EMPTY>
<!ATTLIST Audio
        file CDATA #REQUIRED
>
<!ELEMENT Editorial (#PCDATA | Media | MediaCaption | Sidebar | p | b | i | sub | sup)*>
<!ELEMENT WebLinks (#PCDATA | WebLink | Media | MediaCaption | Link | Measure | p | b | i | sub | sup)*>
<!ELEMENT WebLink (#PCDATA | Media | MediaCaption | Link | Measure | p | b | i | sub | sup)*>
<!ATTLIST WebLink
        nasa (true | false) #REQUIRED
        url CDATA #REQUIRED
        source CDATA #IMPLIED
        title CDATA #REQUIRED
>
<!ELEMENT Glossary (Term*)>
<!ELEMENT Term (#PCDATA)>
<!ATTLIST Term
        name CDATA #REQUIRED
>
<!ELEMENT Pronunciations (Term*)>
<!ELEMENT Translation EMPTY>
<!ATTLIST Translation
        url CDATA #REQUIRED
        kind CDATA #REQUIRED
>
<!ELEMENT p (#PCDATA | b | i | sub | sup | ul | ol | Measure | Link)*>
<!ELEMENT ol (li+)>
<!ELEMENT ul (li+)>
<!ELEMENT li (#PCDATA | i | b | sub | sup | Link)*>
<!ELEMENT i (#PCDATA | b | sub | sup | Measure)*>
<!ELEMENT b (#PCDATA | i | sub | sup | Measure)*>
<!ELEMENT sub (#PCDATA | i | b | sup)*>
<!ELEMENT sup (#PCDATA | i | sub | b)*>
<!ELEMENT Measure (#PCDATA)>
<!ATTLIST Measure
        system (metric | english) "metric"
        metric CDATA #IMPLIED
        english CDATA #IMPLIED
>
```